**Guide Overview**

This section provides detailed documentation on working with the **Vanilla Calendar Pro API**. If you're looking for an introduction to the features, please check out the [«Learn»](https://vanilla-calendar.pro/docs/learn) section.

The Vanilla Calendar Pro API documentation is divided into several functional subsections:

1. **Instance Creation** — how and where to create a calendar instance.
2. **Utilities** — functions that allow you to format dates.
3. **Methods** — available methods for working with the calendar instance.
4. **Settings** — all options that can be provided to change the behavior and display of the calendar.
5. **Actions** — event handlers that allow you to receive and process various interaction data with the calendar.
6. **Popups** — pop-ups allow you to select any day and display brief information about it directly in the calendar when hovering over that day.
7. **Layouts** — templates that allow you to practically alter the entire DOM structure of the calendar and add your own HTML elements.
8. **Styles** — a CSS class object for styling the calendar. It allows you to use any CSS framework, like Tailwind CSS, or custom classes.
9. **Aria-labels** — an object of strings for aria-label. Allows you to localize all calendar labels to ensure accessibility.

Creating an Instance

new Calendar() - creates an instance of Vanilla Calendar Pro, which is an encapsulation of the calendar, its settings, and methods.

If you included Vanilla Calendar Pro using the <script> tag, the object is available as a global variable window.VanillaCalendarPro.

The Calendar instance takes two parameters. The first required parameter can be a CSS selector or HTML element.

The CSS selector or HTML element can represent a wrapper for the calendar, in which the calendar will be initialized, or an «Input».

A calendar wrapper is a <div> tag inside which the calendar itself will be initialized.

Initialization in a calendar wrapper:

html

Copy

<div id="calendar"></div>

ts

Copy

new Calendar('#calendar');

// or

const calendarEl = document.querySelector('#calendar');

new Calendar(calendarEl);

«Input» in the context of this calendar does not necessarily mean an <input> tag; it can be any HTML element, such as a <div>.

When clicking on the «Input», a popup with the calendar will appear.

Initialization in an «Input»:

html

Copy

<input type="text" id="input">

<!-- or -->

<div id="input"></div>

ts

Copy

new Calendar('#input', { input: true });

// or

const calendarInput = document.querySelector('#input');

new Calendar(calendarInput, {

inputMode: true,

});

The second optional parameter is an object defining the settings and actions of the calendar.

ts

Copy

new Calendar('#calendar', {

// Settings

});

**Utilities**

The calendar comes with its utilities, making it easy to work with date formatting.

There are 4 utilities in total, and they are functions that can be used anywhere in your code, even without the calendar.

1. **parseDates(dates: string[])** — Takes an array of date ranges using a delimiter between dates in the string format FormatDateString ('YYYY-MM-DD'). Returns an array of dates in the string format FormatDateString ('YYYY-MM-DD').

tsCopy

import { parseDates } from 'vanilla-calendar-pro/utils';parseDates(['2024-12-12:2024-12-15']); *// return: ['2024-12-12', '2024-12-13', '2024-12-14', '2024-12-15']*

1. **getDateString(date: Date)** — Takes a date of type Date. Returns the date in the string format FormatDateString ('YYYY-MM-DD').

tsCopy

import { getDateString } from 'vanilla-calendar-pro/utils';getDateString(new Date('24.12.2024')); *// return: 2024-12-24*

1. **getDate(date: FormatDateString)** — Takes a date in string format, e.g., FormatDateString ('YYYY-MM-DD'). Returns a date of type Date.

tsCopy

import { getDate } from 'vanilla-calendar-pro/utils';getDate('2024-12-12'); *// return: Tue Dec 24 2024 00:00:00 GMT*

1. **getWeekNumber(date: FormatDateString, weekStartDay: WeekDayID)** — Takes a date in string format FormatDateString ('YYYY-MM-DD') and the week start day, specifically its id of type number from 0 to 6. Returns an object { year: yearNumber, week: weekNumber } for the date specified in the arguments.

tsCopy

import { getWeekNumber } from 'vanilla-calendar-pro/utils';getWeekNumber('2024-12-12', 1); *// return: {year: 2024, week: 50}*

**Methods**

**init()**

The init() method is the main instance method that starts the calendar initialization process.

tsCopy

const calendar = new Calendar(element, params);calendar.init();

**update()**

The update() method allows you to apply new settings to the calendar and perform a reset. This method accepts an object with optional arguments to control the reset, by default resetting the user-selected date, month, and year after the update.

All arguments default to true:

tsCopy

{ year: boolean; month: boolean; dates: boolean | 'only-first'; holidays: boolean; time: boolean;}

* true - will reset to the parameters specified in the settings;
* false - will not perform a reset, leaving the parameters selected by the user;
* 'only-first' - resets all selected dates, leaving only the earliest one. If the date selection type is specified as 'multiple-ranged', a 'mousemove' and 'keydown' handler is added for hovering.

Example usage:

tsCopy

calendar.locale = 'de-AT';calendar.firstWeekday = 0; calendar.update({ dates: true,});

**set()**

If you need to specify new parameters or handlers for a calendar that is not yet initialized or already initialized, you can use the .set() method. This method accepts an object with new parameters and an object with optional arguments to control the reset, by default resetting the user-selected date, month, and year after the update.

Example usage:

tsCopy

calendar.set({ locale: 'de-AT', firstWeekday: 0,}, { dates: true,});

This method can be an alternative to specifying parameters when creating a calendar instance. If you call this method before initialization, do not specify the object for controlling the reset.

tsCopy

const calendar = new Calendar(element);calendar.set({ locale: 'de-AT', firstWeekday: 0 });calendar.init();

**destroy()**

If you need to completely delete the calendar instance, you can use the destroy() method.

tsCopy

calendar.destroy();

**show()**

The show() method allows you to display the calendar if it was hidden.

tsCopy

calendar.show();

**hide()**

The hide() method allows you to hide the calendar if it was shown.

tsCopy

calendar.hide();

**Settings**

**type**

Type: String

Default: 'default'

Options: 'default' | 'multiple' | 'month' | 'year'

tsCopy

new Calendar('#calendar', { type: 'default',});

The type parameter defines the type of calendar displayed.

**inputMode**

Type: Boolean

Default: false

Options: true | false

tsCopy

new Calendar('#calendar', { inputMode: true,});

The inputMode parameter indicates that the mainElement, passed as the first parameter, represents an input field rather than a wrapper for the calendar.

**positionToInput**

Type: String

Default: 'left'

Options: 'auto' | 'center' | 'left' | 'right' | ['bottom' | 'top', 'center' | 'left' | 'right']

tsCopy

new Calendar('#calendar', { positionToInput: 'auto', *// positionToInput: ['bottom', 'center'],*});

This parameter defines the position of the calendar relative to the input if the calendar is initialized with the inputMode parameter.

positionToInput accepts a string with the value 'left', 'center', or 'right', or an array of values [Y-axis, X-axis], where the Y-axis can be 'bottom' or 'top', and the X-axis can be 'left', 'center', or 'right'. If the Y-axis is not specified, the default value 'bottom' is used.

You can use the value positionToInput: 'auto' to automatically determine the best position based on the available space in the viewport. The option allows calculating the available space on all 4 sides and will first try to display the calendar below the input, which is the default position. If there is not enough space below, it will evaluate another best available position.

**firstWeekday**

Type: Number

Default: 1

Options: from 0 to 6

tsCopy

new Calendar('#calendar', { firstWeekday: 1,});

This parameter sets the first day of the week. Specify a number from 0 to 6, where the number represents the day of the week identifier. According to JS standards, the days of the week start with 0, and 0 is Sunday.

**monthsToSwitch**

Type: Number

Default: 1

Options: from 1 to 12

tsCopy

new Calendar('#calendar', { monthsToSwitch: 1,});

The monthsToSwitch parameter controls the number of switchable months.

**themeAttrDetect**

Type: String | False

Default: 'html[data-theme]'

Options: 'string | false

tsCopy

new Calendar('#calendar', { themeAttrDetect: 'html[data-theme]',});

To have the calendar automatically track and apply the site's theme, you can pass a string value in the form of a CSS selector. Square brackets indicate an attribute containing the theme name. By default, the html tag with the data-theme attribute is tracked, but you can configure any other attribute and tag, for example, class, if the class name is used to set the theme: 'html[class]'. If set to false, the theme will be determined by the user's system or the selectedTheme parameter.

**locale**

Type: String

Default: 'en'

Options: Language label | Array<locale>

tsCopy

new Calendar('#calendar', { locale: 'en', *// Or specify an object for your labels // locale: { // months: { // long: [], // short: [], // }, // weekday: { // long: [], // short: [], // } // },*});

This parameter sets the language localization of the calendar. You can specify a language label according to [BCP 47](https://www.iana.org/assignments/language-subtag-registry/language-subtag-registry) or provide arrays of month and weekday names, see more details [here](https://vanilla-calendar.pro/docs/learn/internationalization-locale).

**dateToday**

Type: Date object

Default: 'today'

Options: Date | number | 'YYYY-MM-DD' | 'today'

tsCopy

new Calendar('#calendar', { dateToday: 'today',});

The dateToday parameter defines which day will be considered today for the calendar.

**dateMin**

Type: String

Default: '1970-01-01'

Options: 'Date | number | 'YYYY-MM-DD' | 'today'

tsCopy

new Calendar('#calendar', { dateMin: '1970-01-01',});

The dateMin parameter sets the minimum allowable date that the calendar will consider and which cannot be less than this date.

**dateMax**

Type: String

Default: '2470-12-31'

Options: 'Date | number | 'YYYY-MM-DD' | 'today'

tsCopy

new Calendar('#calendar', { dateMax: '2470-12-31',});

The dateMax parameter sets the maximum allowable date that the calendar will consider and which cannot be greater than this date.

**displayDateMin**

Type: String

Default: '1970-01-01'

Options: 'Date | number | 'YYYY-MM-DD' | 'today'

tsCopy

new Calendar('#calendar', { displayDateMin: '2022-07-01',});

This parameter sets the minimum date that the user can select. Dates earlier than the specified date will be disabled and unavailable for selection.

It is important to note that displayDateMin and displayDateMax disable dates outside the range, while dateMin and dateMax do not create them at all.

**displayDateMax**

Type: String

Default: '2470-12-31'

Options: 'Date | number | 'YYYY-MM-DD' | 'today'

tsCopy

new Calendar('#calendar', { displayDateMax: '2024-07-01',});

This parameter sets the maximum date that the user can select. Dates later than the specified date will be disabled and unavailable for selection.

It is important to note that displayDateMin and displayDateMax disable dates outside the range, while dateMin and dateMax do not create them at all.

**displayDatesOutside**

Type: Boolean

Default: true

Options: true | false

tsCopy

new Calendar('#calendar', { displayDatesOutside: false,});

With this parameter, you can decide whether to display days from the previous and next month.

**displayDisabledDates**

Type: Boolean

Default: false

Options: true | false

tsCopy

new Calendar('#calendar', { displayDisabledDates: false,});

This parameter determines whether all days, including disabled days, will be displayed.

**displayMonthsCount**

Type: Number

Default: 2

Options: from 2 to 12

tsCopy

new Calendar('#calendar', { displayMonthsCount: 2,});

The displayMonthsCount parameter defines the number of months displayed if the calendar type is set to 'multiple'.

**disableDates**

Type: String[] | Number[] | Date[]

Default: null

Options: ['YYYY-MM-DD'] | [Number] | [Date] | null

tsCopy

new Calendar('#calendar', { disableDates: ['2022-08-10:2022-08-15', '2022-08-20', 1722152977141, new Date()],});

This parameter allows you to disable specified dates, regardless of the specified range.

To specify a date range, use any delimiter between dates within a single string.

**disableAllDates**

Type: Boolean

Default: false

Options: true | false

tsCopy

new Calendar('#calendar', { disableAllDates: true,});

This parameter disables all days and can be useful when using enableDates.

**disableDatesPast**

Type: Boolean

Default: false

Options: true | false

tsCopy

new Calendar('#calendar', { disableDatesPast: true,});

This parameter disables all past days.

**disableDatesGaps**

Type: Boolean

Default: false

Options: true | false

tsCopy

new Calendar('#calendar', { disableDatesGaps: true,});

This parameter disables the selection of days within a range with disabled dates. It only works if the selectionDatesMode parameter is set to 'multiple-ranged'.

**disableWeekdays**

Type: Number

Default: []

Options: from 0 to 6

tsCopy

new Calendar('#calendar', { disableWeekdays: [0, 6],});

This parameter allows you to disable specified weekdays. Specify an array with numbers from 0 to 6, where each number represents a day of the week identifier. According to JS standards, the days of the week start with 0, and 0 is Sunday.

**disableToday**

Type: Boolean

Default: false

Options: true | false

tsCopy

new Calendar('#calendar', { disableToday: true,});

With this parameter, you can disable the selection of today's date in the calendar.

**enableDates**

Type: String[] | Number[] | Date[]

Default: null

Options: ['YYYY-MM-DD'] | [Number] | [Date] | null

tsCopy

new Calendar('#calendar', { enableDates: ['2022-08-11:2022-08-16', '2022-08-20', 1722152977141, new Date()],});

This parameter allows you to enable specified dates, regardless of the range and disabled dates.

To specify a date range, use any delimiter between dates within a single string.

**enableEdgeDatesOnly**

Type: Boolean

Default: true

Options: true | false

tsCopy

new Calendar('#calendar', { enableEdgeDatesOnly: true,});

This parameter allows you to get only the start and end dates selected by the user, ignoring intermediate dates. This parameter only works if selectionDatesMode is set to 'multiple-ranged'.

It is important to note that when using this parameter, disabled dates within the date range will have no effect. Therefore, use this option only if you are interested in the start and end dates selected by the user.

**enableDateToggle**

Type: Boolean | Function

Default: true

Options: true | false | () => false

tsCopy

new Calendar('#calendar', { enableDateToggle: false, *// or with a callback* enableDateToggle: (*self*) => (new Date(self.selectedDates[0]) < new Date()),});

If the enableDateToggle parameter is true or the callback returns true, then clicking on a selected date again will deselect it.

**enableWeekNumbers**

Type: Boolean

Default: false

Options: true | false

tsCopy

new Calendar('#calendar', { enableWeekNumbers: true,});

With this parameter, you can decide whether to display week numbers in the year.

**enableMonthChangeOnDayClick**

Type: Boolean

Default: true

Options: true | false

tsCopy

new Calendar('#calendar', { enableMonthChangeOnDayClick: false,});

With this parameter, you can decide whether the month will switch when clicking on a day from the previous or next month.

**enableJumpToSelectedDate**

Type: Boolean

Default: false

Options: true | false

tsCopy

new Calendar('#calendar', { enableJumpToSelectedDate: true, selectedDates: ['2018-05-02'],});

If this option is enabled and one or more selected dates are specified, but without specifying selectedMonth and selectedYear, the calendar will jump to the first selected date. If set to false, the calendar will always open for the current month and year.

This option has no effect if selectedMonth and selectedYear are specified.

**selectionDatesMode**

Type: String | false

Default: 'single'

Options: 'single' | 'multiple' | 'multiple-ranged' | false

tsCopy

new Calendar('#calendar', { selectionDatesMode: 'single',});

This parameter determines whether selecting one or multiple days is allowed, or if date selection is completely disabled.

**selectionMonthsMode**

Type: Boolean

Default: true

Options: true | false | 'only-arrows'

tsCopy

new Calendar('#calendar', { selectionMonthsMode: false,});

This parameter allows you to disable month selection, allow month switching only with arrows, or allow month switching in any way.

**selectionYearsMode**

Type: Boolean

Default: true

Options: true | false | 'only-arrows'

tsCopy

new Calendar('#calendar', { selectionYearsMode: false,});

This parameter allows you to disable year selection, allow year switching only with arrows, or allow year switching in any way.

**selectionTimeMode**

Type: False | Number

Default: false

Options: false | 24 | 12

tsCopy

new Calendar('#calendar', { selectionTimeMode: true,});

This parameter enables time selection. You can also specify the time format using a number: 24-hour or 12-hour format.

**selectedDates**

Type: String[] | Number[] | Date[]

Default: null

Options: ['YYYY-MM-DD'] | [Number] | [Date] | null

tsCopy

new Calendar('#calendar', { selectedDates: ['2022-08-10:2022-08-15', '2022-08-20', 1722152977141, new Date()],});

This parameter allows you to specify a list of dates that will be selected when the calendar is initialized.

To specify a date range, use any delimiter between dates within a single string.

**selectedMonth**

Type: Number

Default: null

Options: from 0 to 11 | null

tsCopy

new Calendar('#calendar', { selectedMonth: 0,});

This parameter defines the month that will be displayed when the calendar is initialized. According to JS standards, months are numbered from 0 to 11.

**selectedYear**

Type: Number

Default: null

Options: Number (YYYY) | null

tsCopy

new Calendar('#calendar', { selectedYear: 2022,});

This parameter defines the year that will be displayed when the calendar is initialized.

**selectedHolidays**

Type: String[] | Number[] | Date[]

Default: null

Options: ['YYYY-MM-DD'] | [Number] | [Date] | null

tsCopy

new Calendar('#calendar', { selectedHolidays: ['2022-08-10:2022-08-15', '2022-08-20', 1722152977141, new Date()],});

This parameter allows you to specify dates that will be considered holidays and will receive an additional data attribute for styling.

To specify a date range, use any delimiter between dates within a single string.

**selectedWeekends**

Type: Number

Default: [0, 6]

Options: number[0-6]

tsCopy

new Calendar('#calendar', { selectedWeekends: [0, 6],});

This parameter allows you to specify the weekend days of the week. Specify an array with numbers from 0 to 6, where each number represents a day of the week identifier. According to JS standards, the days of the week start with 0, and 0 is Sunday.

**selectedTime**

Type: String

Default: null

Options: 'hh:mm aa' | null

tsCopy

new Calendar('#calendar', { selectedTime: '03:44 AM',});

This parameter allows you to set the time that will be displayed when the calendar is initialized. The time is set in the format 'hh:mm aa', where 'aa' is the AM/PM marker. If using the 24-hour format, the 'aa' marker is not required.

**selectedTheme**

Type: String

Default: 'system'

Options: string (custom theme) | 'light' | 'dark' | 'system'

tsCopy

new Calendar('#calendar', { selectedTheme: 'system',});

This parameter defines the theme of the calendar. By default, the theme is determined by the user's system or the site settings.

**timeMinHour**

Type: Number

Default: 0

Options: from 0 to 23

tsCopy

new Calendar('#calendar', { timeMinHour: 0,});

This parameter specifies which hour will be the minimum for selection.

**timeMaxHour**

Type: Number

Default: 23

Options: from 0 to 23

tsCopy

new Calendar('#calendar', { timeMaxHour: 23,});

This parameter specifies which hour will be the maximum for selection.

**timeMinMinute**

Type: Number

Default: 0

Options: from 0 to 59

tsCopy

new Calendar('#calendar', { timeMinMinute: 0,});

This parameter specifies which minute will be the minimum for selection.

**timeMaxMinute**

Type: Number

Default: 59

Options: from 0 to 59

tsCopy

new Calendar('#calendar', { timeMaxMinute: 59,});

This parameter specifies which minute will be the maximum for selection.

**timeControls**

Type: String

Default: 'all'

Options: 'all' | 'range'

tsCopy

new Calendar('#calendar', { timeControls: 'all',});

This parameter defines the method of time selection: 'all' (any method) or 'range' (only with the controller).

**timeStepHour**

Type: Number

Default: 1

Options: from 1 to 23

tsCopy

new Calendar('#calendar', { timeStepHour: 1,});

This parameter sets the step for the hour controller.

**timeStepMinute**

Type: Number

Default: 1

Options: from 1 to 59

tsCopy

new Calendar('#calendar', { timeStepMinute: 1,});

This parameter sets the step for the minute controller.

**sanitizerHTML**

Type: Function

Default: (html) => html

tsCopy

import DOMPurify from 'dompurify'; new Calendar('#calendar', { sanitizerHTML: (*html*) => DOMPurify.sanitize(html),});

sanitizerHTML can sanitize HTML templates to make them safe for CSP.

Note that the example uses the third-party library [dompurify](https://www.npmjs.com/package/dompurify" \t "_blank). sanitizerHTML is not required for the calendar to function.

**Actions**

**onClickDate()**

Type: Function

Default: null

Options: onClickDate(self, event) => void | null

tsCopy

new Calendar('#calendar', { onClickDate(*self*, *event*) {},});

This method is triggered after clicking on a day in the calendar. You can get the following parameters:

* self - reference to the initialized calendar;
* event - mouse event.

It is important to know that each HTML day element contains a data attribute with the full date in the format YYYY-MM-DD. If you need to get the day, month, and year separately, you can use standard JS methods. For example: new Date('2022-11-07').getDate() will return 7.

**onClickWeekDay()**

Type: Function

Default: null

Options: onClickWeekDay(self, day, dateEls, event) => void | null

tsCopy

new Calendar('#calendar', { onClickWeekDay(*self*, *day*, *dateEls*, *event*) {},});

This method is triggered after clicking on a weekday in the calendar. You can get the following parameters:

* self - reference to the initialized calendar;
* day - week day;
* dateEls - array of days (html elements);
* event - mouse event.

**onClickWeekNumber()**

Type: Function

Default: null

Options: onClickWeekNumber(self, number, year, dateEls, event) => void | null

tsCopy

new Calendar('#calendar', { onClickWeekNumber(*self*, *number*, *year*, *dateEls*, *event*) {},});

This method is triggered after clicking on a week number in the calendar, but for it to work, the enableWeekNumbers parameter must be set to true. You can get the following parameters:

* self - reference to the initialized calendar;
* number - week number;
* year - year of the week;
* dateEls - array of days (html elements);
* event - mouse event.

**onClickTitle()**

Type: Function

Default: null

Options: onClickTitle(self, event) => void | null

tsCopy

new Calendar('#calendar', { onClickTitle(*self*, *event*) {},});

This method is triggered after clicking on the month or year title in the calendar. You can get the following parameters:

* self - reference to the initialized calendar;
* event - mouse event.

**onClickMonth()**

Type: Function

Default: null

Options: onClickMonth(self, event) => void | null

tsCopy

new Calendar('#calendar', { onClickMonth(*self*, *event*) {},});

This method is triggered after selecting a month in the calendar. You can get the following parameters:

* self - reference to the initialized calendar;
* event - mouse event.

**onClickYear()**

Type: Function

Default: null

Options: onClickYear(self, event) => void | null

tsCopy

new Calendar('#calendar', { onClickYear(*self*, *event*) {},});

This method is triggered after selecting a year in the calendar. You can get the following parameters:

* self - reference to the initialized calendar;
* event - mouse event.

**onClickArrow()**

Type: Function

Default: null

Options: onClickArrow(self, event) => void | null

tsCopy

new Calendar('#calendar', { onClickArrow(*self*, *event*) {},});

This method is triggered after clicking on an arrow in the calendar. You can get the following parameters:

* self - reference to the initialized calendar;
* event - mouse event.

**onChangeTime()**

Type: Function

Default: null

Options: onChangeTime(self, event, isError) => void | null

tsCopy

new Calendar('#calendar', { onChangeTime(*self*, *event*) {},});

This method is triggered after changing the time in the calendar. You can get the following parameters:

* self - reference to the initialized calendar;
* event - change event;
* isError - returns true if the user entered an incorrect time.

**onChangeToInput()**

Type: Function

Default: null

Options: onChangeToInput(self, event) => void | null

tsCopy

new Calendar('#calendar', { onChangeToInput(*self*, *event*) {},});

For this method to work, the inputMode parameter must be set to true. This method is triggered after clicking on a day in the calendar or changing the time in any way. You can get the following parameters:

* self - reference to the initialized calendar;
* event - event.

**onCreateDateRangeTooltip()**

Type: Function

Default: null

Options: onCreateDateRangeTooltip(self, dateEl, tooltipEl, dateElBCR, mainElBCR) => void | null

tsCopy

new Calendar('#calendar', { onCreateDateRangeTooltip(*self*, *dateEl*, *tooltipEl*, *dateElBCR*, *mainElBCR*) {},});

Allows creating a tooltip for a date range. Triggers on clicking and hovering over a day if the selectionDatesMode parameter is set to 'multiple-ranged'. You can get the following parameters:

* self - reference to the initialized calendar.
* dateEl - HTML date element;
* tooltipEl - HTML tooltip element;
* dateElBCR - object with information about the position and size of the HTML date element;
* mainElBCR - object with information about the position and size of the main HTML calendar element.

**onCreateDateEls()**

Type: Function

Default: null

Options: onCreateDateEls(self, dateEl) => void | null

tsCopy

new Calendar('#calendar', { onCreateDateEls(*self*, *dateEl*) {},});

This method is triggered during calendar initialization and any changes. It provides access to information about each day. You can get the following parameters:

* self - reference to the initialized calendar;
* dateEl - HTML date element.

**onCreateMonthEls()**

Type: Function

Default: null

Options: onCreateMonthEls(self, monthEl) => void | null

tsCopy

new Calendar('#calendar', { onCreateMonthEls(*self*, *monthEl*) {},});

This method is triggered when the calendar type is set to 'month'. The calendar type also becomes 'month' when the user clicks on the month title or during initialization with the parameter type = 'month'. It provides access to information about each month. You can get the following parameters:

* self - reference to the initialized calendar;
* monthEl - HTML month element.

**onCreateYearEls()**

Type: Function

Default: null

Options: onCreateYearEls(self, yearEl) => void | null

tsCopy

new Calendar('#calendar', { onCreateYearEls(*self*, *yearEl*) {},});

This method is triggered when the calendar type is set to 'year'. The calendar type becomes 'year' when the user clicks on the year title or during initialization with the parameter type = 'year'. It provides access to information about each year. You can get the following parameters:

* self - reference to the initialized calendar;
* yearEl - HTML year element.

**onInit()**

Type: Function

Default: null

Options: onInit(self) => void | null

tsCopy

new Calendar('#calendar', { onInit(*self*) {},});

This method is triggered during calendar initialization. If the inputMode parameter is set to true, the method will execute on the first display of the calendar, as this is when the calendar is initialized.

* self - reference to the initialized calendar.

**onUpdate()**

Type: Function

Default: null

Options: onUpdate(self) => void | null

tsCopy

new Calendar('#calendar', { onUpdate(*self*) {},});

This method is triggered when the calendar is updated/reset using the .update() method.

* self - reference to the initialized calendar.

**onDestroy()**

Type: Function

Default: null

Options: onDestroy(self) => void | null

tsCopy

new Calendar('#calendar', { onDestroy(*self*) {},});

This method is triggered when the calendar is destroyed.

* self - reference to the initialized calendar.

**onShow()**

Type: Function

Default: null

Options: onShow(self) => void | null

tsCopy

new Calendar('#calendar', { onShow(*self*) {},});

This method is triggered when the calendar is displayed to the user, but only if the inputMode parameter is set to true.

* self - reference to the initialized calendar.

**onHide()**

Type: Function

Default: null

Options: onHide(self) => void | null

tsCopy

new Calendar('#calendar', { onHide(*self*) {},});

This method is triggered when the calendar is hidden, but only if the inputMode parameter is set to true.

* self - reference to the initialized calendar.

Popups

Popups allow you to highlight any day and display brief information about it directly in the calendar when hovering over the day.

popups['date']

Type: String

Default: null

Options: 'YYYY-MM-DD': | null

ts

Copy

new Calendar('#calendar', {

popups: {

'2022-06-28': {},

}

});

Dates in the format YYYY-MM-DD are used as keys. In the given example, a popup is set for June 28, 2022.

popups['date'].modifier

Type: String

Default: null

Options: CSS classes | null

ts

Copy

new Calendar('#calendar', {

popups: {

'2022-06-28': {

modifier: 'bg-red color-pink',

},

}

});

modifier accepts arbitrary CSS classes separated by spaces. Using these classes, you can style the date to make it highlighted or change its appearance.

popups['date'].html

Type: String

Default: null

Options: '' | HTML | null

ts

Copy

new Calendar('#calendar', {

popups: {

'2022-06-28': {

modifier: 'bg-red color-pink',

html: `<div>

<u><b>12:00 PM</b></u>

<p style="margin: 5px 0 0;">Airplane in Las Vegas</p>

</div>`,

// or just text

// html: 'Airplane in Las Vegas',

},

}

});

html accepts plain text or HTML markup for formatting the popup. In this example, when hovering over June 28, 2022, a popup will be displayed with the text "Airplane in Las Vegas" and the time "12:00 PM", and the styles specified in the classes bg-red and color-pink will be applied.

**Layouts**

Layouts allow you to almost completely change the DOM structure of the calendar and add your own HTML elements, such as buttons. Each type of calendar has its own default template, and you can customize each of them.

Tags containing the symbol **«#»** are registered components of the calendar and should contain a closing slash at the end of the tag, except for the tag **<#Multiple><#/Multiple>**, which wraps one month. All default templates list all possible components for that template.

**layouts.default**

Type: String

Default: string

Options: string

tsCopy

new Calendar('#calendar', { layouts: { default: ` <div class="${self.styles.header}" data-vc="header" role="toolbar" aria-label="${self.labels.navigation}"> <#ArrowPrev [month] /> <div class="${self.styles.headerContent}" data-vc-header="content"> <#Month /> <#Year /> </div> <#ArrowNext [month] /> </div> <div class="${self.styles.wrapper}" data-vc="wrapper"> <#WeekNumbers /> <div class="${self.styles.content}" data-vc="content"> <#Week /> <#Dates /> <#DateRangeTooltip /> </div> </div> <#ControlTime /> ` }});

This is the default template for displaying one month and its dates.

**layouts.multiple**

Type: String

Default: string

Options: string

tsCopy

new Calendar('#calendar', { layouts: { multiple: ` <div class="${self.styles.controls}" data-vc="controls" role="toolbar" aria-label="${self.labels.navigation}"> <#ArrowPrev [month] /> <#ArrowNext [month] /> </div> <div class="${self.styles.grid}" data-vc="grid"> <#Multiple> <div class="${self.styles.column}" data-vc="column" role="region"> <div class="${self.styles.header}" data-vc="header"> <div class="${self.styles.headerContent}" data-vc-header="content"> <#Month /> <#Year /> </div> </div> <div class="${self.styles.wrapper}" data-vc="wrapper"> <#WeekNumbers /> <div class="${self.styles.content}" data-vc="content"> <#Week /> <#Dates /> </div> </div> </div> <#/Multiple> <#DateRangeTooltip /> </div> <#ControlTime /> ` }});

This is the default template for displaying multiple months and their dates.

**layouts.month**

Type: String

Default: string

Options: string

tsCopy

new Calendar('#calendar', { layouts: { month: ` <div class="${self.styles.header}" data-vc="header" role="toolbar" aria-label="${self.labels.navigation}"> <div class="${self.styles.headerContent}" data-vc-header="content"> <#Month /> <#Year /> </div> </div> <div class="${self.styles.wrapper}" data-vc="wrapper"> <div class="${self.styles.content}" data-vc="content"> <#Months /> </div> </div> ` }});

This is the default template for selecting a month.

**layouts.year**

Type: String

Default: string

Options: string

tsCopy

new Calendar('#calendar', { layouts: { year: ` <div class="${self.styles.header}" data-vc="header" role="toolbar" aria-label="${self.labels.navigation}"> <#ArrowPrev [year] /> <div class="${self.styles.headerContent}" data-vc-header="content"> <#Month /> <#Year /> </div> <#ArrowNext [year] /> </div> <div class="${self.styles.wrapper}" data-vc="wrapper"> <div class="${self.styles.content}" data-vc="content"> <#Years /> </div> </div> ` }});

This is the default template for selecting a year.

**Styles**

styles provides the ability to override any CSS class in the calendar. You can replace any values with a list of CSS classes.

Below is a list of all default classes.

tsCopy

new Calendar('#calendar', { styles: { calendar: 'vc', controls: 'vc-controls', grid: 'vc-grid', column: 'vc-column', header: 'vc-header', headerContent: 'vc-header\_\_content', month: 'vc-month', year: 'vc-year', arrowPrev: 'vc-arrow vc-arrow\_prev', arrowNext: 'vc-arrow vc-arrow\_next', wrapper: 'vc-wrapper', content: 'vc-content', months: 'vc-months', monthsMonth: 'vc-months\_\_month', years: 'vc-years', yearsYear: 'vc-years\_\_year', week: 'vc-week', weekDay: 'vc-week\_\_day', weekNumbers: 'vc-week-numbers', weekNumbersTitle: 'vc-week-numbers\_\_title', weekNumbersContent: 'vc-week-numbers\_\_content', weekNumber: 'vc-week-number', dates: 'vc-dates', date: 'vc-date', dateBtn: 'vc-date\_\_btn', datePopup: 'vc-date\_\_popup', dateRangeTooltip: 'vc-date-range-tooltip', time: 'vc-time', timeContent: 'vc-time\_\_content', timeHour: 'vc-time\_\_hour', timeMinute: 'vc-time\_\_minute', timeKeeping: 'vc-time\_\_keeping', timeRanges: 'vc-time\_\_ranges', timeRange: 'vc-time\_\_range', },});

**Aria Labels**

labels provides the ability to localize all aria-labels in the calendar for accessibility.

Below is a list of all default aria-labels.

tsCopy

new Calendar('#calendar', { labels: { application: 'Calendar', navigation: 'Calendar Navigation', arrowNext: { month: 'Next month', year: 'Next list of years', }, arrowPrev: { month: 'Previous month', year: 'Previous list of years', }, month: 'Select month, current selected month:', months: 'List of months', year: 'Select year, current selected year:', years: 'List of years', week: 'Days of the week', weekNumber: 'Numbers of weeks in a year', dates: 'Dates in the current month', selectingTime: 'Selecting a time ', inputHour: 'Hours', inputMinute: 'Minutes', rangeHour: 'Slider for selecting hours', rangeMinute: 'Slider for selecting minutes', btnKeeping: 'Switch AM/PM, current position:', },});

**Introduction to Vanilla Calendar Pro**

**Vanilla Calendar Pro** is a powerful, flexible, and lightweight tool for handling dates and times, created for developers who need a functional and easily customizable calendar for web applications or websites. It is independent of external libraries and highly performant, making it an excellent choice for integrating into any projects that require a calendar.

This calendar is designed for developers working on a wide variety of projects, whether personal sites, corporate portals, or complex web applications. Vanilla Calendar Pro is perfect for those looking for a simple date display solution and those needing more advanced features like time selection and interactive actions.

**Key Features**

Vanilla Calendar Pro offers a rich set of features that allow the creation of convenient and adaptive calendar widgets.

Key features include:

* **Lightweight**: The final JavaScript file is minified and optimized for fast loading.
* **Dependency-Free**: Completely standalone, with no need for additional libraries.
* **Easy Localization**: Supports easy localization for any language.
* **Customizable**: Easily configurable through CSS and HTML markup.
* **Multiple Instances**: Allows unlimited calendars on a single page.
* **Theme Support**: Automatically switches between light and dark themes and supports custom themes.
* **Week Start Customization**: Enables choosing any day of the week as the starting day.
* **Weekend Customization**: Allows setting custom weekends for each week.
* **Week Number Display**: Can display week numbers throughout the year.
* **Not Tied to <input>**: Unlike many calendars, it is not limited to use with the <input> element.
* **Accessibility**: Includes ARIA labels, tabindex, and full keyboard navigation, enhancing accessibility.
* **Date and Time Range Selection**: Supports selecting date and time ranges with minimum and maximum limits.
* **Pop-Ups and Tooltips**: Allows setting up pop-ups with custom information and adds tooltips for date range selections.

**Try Vanilla Calendar Pro**

Below is a live example of Vanilla Calendar Pro in a JS sandbox. You can modify the parameters and instantly see how the calendar adapts to your settings.

import { Calendar } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const calendar = new Calendar('#calendar');

calendar.init();i

**Installation and Usage**

Vanilla Calendar Pro is easily integrated into any project. There are several installation methods, depending on how you prefer to manage dependencies and build your project.

**Installation via Package Manager**

The most common way to install Vanilla Calendar Pro is by using a package manager. This method is ideal for projects using Node.js and modern build tools.

1. Install the package:

bashCopy

npm install vanilla-calendar-pro*# or*yarn add vanilla-calendar-pro*# or*pnpm add vanilla-calendar-pro

1. Create an HTML element in the body of your document with an arbitrary CSS selector:

htmlCopy

<html> <head> </head> <body> <div id="calendar"></div> </body></html>

For demonstration purposes in this section, we will use #calendar as the CSS selector, but you can create and use any other selector.

1. Import the script, create a calendar instance, and initialize it in your JavaScript or TypeScript file.

tsCopy

import { Calendar } from 'vanilla-calendar-pro'; const calendar = new Calendar('#calendar', { *// Your settings*});calendar.init();

1. Import the styles in the same file. The index.css file contains the layout grid for the calendar, as well as light and dark themes.

tsCopy

import 'vanilla-calendar-pro/styles/index.css';

You also have the option to include the layout and theme styles separately, like this:

tsCopy

import 'vanilla-calendar-pro/styles/layout.css'; *// Only the skeleton*import 'vanilla-calendar-pro/styles/themes/light.css'; *// Light theme*import 'vanilla-calendar-pro/styles/themes/dark.css'; *// Dark theme// or any other custom theme...*

1. Full example of simple initialization without any custom settings:

**TS Live Example**

Refresh

Top of Form

Bottom of Form

Open

import { Calendar } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const calendar = new Calendar('#calendar');

calendar.init();

As you may have noticed in this example, we are using a flat calendar view without using the **«Input»** field, if you are interested in how you can integrate a calendar into **«Input»**, check out [this example](https://vanilla-calendar.pro/docs/learn/type-default#with-input).

**Local or CDN**

If you need to quickly integrate Vanilla Calendar Pro without using build tools or package managers, you can include it via CDN or [download the archive](https://cdn.jsdelivr.net/npm/vanilla-calendar-pro@latest/package.zip) with the latest version and include it locally.

htmlCopy

<html> <head> <link href="https://cdn.jsdelivr.net/npm/vanilla-calendar-pro/styles/index.css" rel="stylesheet"> <script src="https://cdn.jsdelivr.net/npm/vanilla-calendar-pro/index.js" defer></script> </head> <body style="display: flex; align-items: start"> <div id="calendar"></div> <script> document.addEventListener('DOMContentLoaded', () => { *// Destructure the Calendar constructor* const { Calendar } = window.VanillaCalendarPro; *// Create a calendar instance and initialize it.* const calendar = new Calendar('#calendar'); calendar.init(); }); </script> </body></html>

**Calendar Types**

**Default (Single)**

**Static**

The 'default' calendar type displays one month, allows you to select days, navigate between months using arrows, and select the month and year from the respective headers. This is the standard display mode for the calendar.

**TS Live Example**

Refresh

Top of Form

Bottom of Form

Open

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

type: 'default',

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**With Input**

If you need to display the calendar when clicking on an **«Input»**, you can easily configure it by initializing it with the inputMode: true parameter.

It is important to note that **«Input»** in the context of this calendar does not necessarily have to be an <input> tag. It can be any HTML element, such as a <div>. In **«Input»**, you can initialize any type of calendar.

By default, the calendar does not write any values to the **«Input»** field, giving you unique control over what you want to see in the value.

**TS Live Example**

Refresh

Top of Form

Bottom of Form

Open

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

inputMode: true,

positionToInput: 'auto',

onChangeToInput(self) {

if (!self.context.inputElement) return;

if (self.context.selectedDates[0]) {

self.context.inputElement.value = self.context.selectedDates[0];

*// if you want to hide the calendar after picking a date*

self.hide();

} else {

self.context.inputElement.value = '';

}

},

};

const calendarInput = new Calendar('#calendar', options);

calendarInput.init();

**Calendar Types**

**Month**

The 'month' calendar type displays a list of months and allows the user to select months and years from the respective headers. This mode is useful if you need to restrict the user's selection to only the month and year, without the ability to select specific days.

**TS Live Example**

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

type: 'month',

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Calendar Types**

**Multiple**

The 'multiple' calendar type displays multiple months, allowing you to select days in each of them. This type of calendar is useful when the user needs to select multiple dates across different months. To do this, you need to use the selectionDatesMode parameter and set its value to 'multiple'.

Example code for creating a calendar with the 'multiple' type:

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

type: 'multiple',

displayMonthsCount: 2,

monthsToSwitch: 1,

selectionDatesMode: 'multiple',

};

const calendar = new Calendar('#calendar', options);

calendar.init();

If you need to select date ranges, you can use the selectionDatesMode parameter and set its value to 'multiple-ranged'. This allows you to select date ranges instead of individual days.

When the selectionDatesMode parameter is set to 'multiple-ranged', for performance optimization, the array of selected dates contains only the start and end dates. You can disable this and get the full list of selected dates using enableEdgeDatesOnly.

**TS Live Example**

Refresh

Top of Form

Bottom of Form

Open

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

type: 'multiple',

displayMonthsCount: 2,

monthsToSwitch: 2,

displayDatesOutside: false,

disableDatesPast: true,

enableEdgeDatesOnly: true,

selectionDatesMode: 'multiple-ranged',

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Calendar Types**

**Year**

The 'year' calendar type displays a list of years, allowing the user to select a year from the list and a month from the corresponding header. This mode is useful if you need to restrict the user's selection to only the year and month, excluding the ability to select specific days.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

type: 'year',

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Internationalization**

**Localization**

If your locale is supported by the [.toLocaleString()](https://developer.mozilla.org/en-US/docs/Web/JavaScript/Reference/Global_Objects/Date/toLocaleString) method, you can simply pass it to the locale parameter to localize the calendar.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

locale: 'de-AT', // Austrian-German

};

const calendar = new Calendar('#calendar', options);

calendar.init();

If the locale is not supported or translated incorrectly, you can always set the locale manually. To do this, you need to provide arrays of month and weekday names instead of the language tag.

**TS Live Example**

Refresh

Top of Form

Bottom of Form

Open

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

locale: {

months: {

short: ['Vör', 'Thors', 'Skadi', 'Freya', 'Baldur', 'Njord', 'Tyr', 'Frigg', 'Odin', 'Loki', 'Hel', 'Idunn'],

long: [

'Vörmánuðr',

'Thorsmánuðr',

'Skadimánuðr',

'Freymánuðr',

'Baldurmánuðr',

'Njordmánuðr',

'Tyrmánuðr',

'Friggmánuðr',

'Odinmánuðr',

'Lokimánuðr',

'Helmánuðr',

'Idunnmánuðr',

],

},

weekdays: {

short: ['Sunna', 'Mani', 'Tiw', 'Woden', 'Thor', 'Frigg', 'Saturn'],

long: ['Sunnandæg', 'Manadæg', 'Tiwesdæg', 'Wodensdæg', 'Thorsdæg', 'Friggsdæg', 'Saturnsdag'],

},

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**nternationalization**

**Week Numbers**

In some countries, week numbers are used to denote dates. You can enable the display of week numbers in the calendar by setting the enableWeekNumbers parameter to true.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

enableWeekNumbers: true,

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Internationalization**

**First Day of the Week and Weekends**

By default, the calendar is based on the European standard **ISO 8601**. This means that the first day of the week is Monday.

Using separate parameters to define the first day of the week and the displayed weekends, you can specify any day as the first day of the week and assign any days of the week as weekends or completely disable them by specifying an empty array.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

firstWeekday: 0,

selectedWeekends: [0, 3, 6],

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Internationalization**

**Additional Weekends and Holidays**

In the calendar, you can specify additional weekends or holidays that will be marked in red. These days should be set manually.

**TS Live Example**

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectedMonth: 0,

selectedYear: 2022,

selectedHolidays: ['2022-01-01:2022-01-05', '2022-01-10', '2022-01-13'],

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Managing Dates and Time**

**Maximum and Minimum Date**

The date range in the calendar can be set using the dateMin and dateMax parameters. These parameters specify the allowed range of dates in the calendar.

By default, the minimum date is '1970-01-01', which corresponds to the beginning of [UNIX time](https://en.wikipedia.org/wiki/Unix_time). The maximum date is set to '2470-12-31' by default and is chosen arbitrarily.

If you need to set a specific range of possible dates, replace the values of the dateMin and dateMax parameters with the dates you need. Note that the calendar will not process dates outside the specified range.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

dateMin: '1920-01-01',

dateMax: '2038-12-31',

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Managing Dates and Time**

**Display Date Range**

The displayDateMin and displayDateMax parameters define the range of dates that can be displayed in the calendar but do not affect the calendar's lifecycle. They only indicate which dates are allowed to be displayed and selected.

For example, if the displayDisabledDates parameter is set to true, then the minimum and maximum years available for viewing by the user will be determined by the values of the dateMin and dateMax parameters.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

dateMin: '1920-01-01',

dateMax: '2038-12-31',

displayDateMin: '2000-01-01',

displayDateMax: '2024-12-31',

displayDisabledDates: false,

};

const calendar = new Calendar('#calendar', options);

calendar.init();

Changing the displayDisabledDates parameter allows you to control the dates available for viewing and selection in the calendar.

**Managing Dates and Time**

**Enable or Disable Days**

You may need to disable certain days so that they are not available for selection.

**TS Live Example**

Refresh

Top of Form

Bottom of Form

Open

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

displayDateMin: '2022-07-01',

displayDateMax: '2022-09-30',

disableDates: ['2022-08-10:2022-08-13', '2022-08-22'],

selectedYear: 2022,

selectedMonth: 7,

};

const calendar = new Calendar('#calendar', options);

calendar.init();

Sometimes it may be easier to disable all days and enable specific days rather than listing the disabled days.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

disableAllDates: true,

enableDates: ['2022-08-10:2022-08-13', '2022-08-22'],

selectedYear: 2022,

selectedMonth: 7,

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Managing Dates and Time**

**Enable Time Selection**

By default, time selection is disabled, but you can easily enable it and configure it according to your needs.

**12-Hour Day with AM/PM**

You can enable the 12-hour time format and add AM/PM markers.

TSL Live Example   
  
import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectionTimeMode: 12,

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**24-Hour Day**

If you need a 24-hour time format without AM/PM, you can configure it as follows.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectionTimeMode: 24,

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Setting Your Own Time**

You can set the initial time when initializing the calendar. For a 24-hour day, you do not need to specify the AM/PM marker.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectionTimeMode: 12,

selectedTime: '03:44 AM',

};

const calendar = new Calendar('#calendar', options);

calendar.init();  
**Managing Time Range**

You can set the possible time range.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectionTimeMode: 12,

timeMinHour: 6,

timeMaxHour: 21,

timeMinMinute: 10,

timeMaxMinute: 40,

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Managing Time Step**

In addition to everything else, you can configure the time step for minutes and hours. You can also disable the ability to manually enter time in the input field.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectionTimeMode: 12,

timeControls: 'range',

timeStepHour: 5,

timeStepMinute: 5,

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Managing Dates and Time**

**Disable Day, Month, and Year Selection**

The calendar allows you to easily disable the ability to select the day, month, or year individually.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectionDatesMode: false,

selectionMonthsMode: false,

selectionYearsMode: false,

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Managing Dates and Time**

**Custom Today**

The calendar provides the ability to specify which day should be considered today.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

dateToday: new Date('2022-01-07'),

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Managing Dates and Time**

**Selected Days, Month, and Year on Initialization**

The calendar allows you to explicitly specify selected days upon initialization, as well as the month and year that will be displayed regardless of the current date.

This is useful if you need to pre-select certain dates and set a specific month and year.

**TS Live Example**

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectionDatesMode: 'multiple',

selectedDates: ['2022-01-09:2022-01-13', '2022-01-22'],

selectedMonth: 0,

selectedYear: 2022,

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Action Handlers**

**Handling Day Click**

For user interaction with the calendar, various actions are provided, one of which is onClickDate(). This action allows you to track when a user clicks on a specific day in the calendar.

Example with outputting the selected day to the console:

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

onClickDate(self) {

console.log(self.context.selectedDates);

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

Note that the selected day is represented as an array, as the user can select not only a single day but also a range of dates if allowed by the calendar parameters.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectionDatesMode: 'multiple-ranged',

onClickDate(self) {

console.log(self.context.selectedDates);

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Handling Month Click in Month List**

When a month is clicked in the list of all months, you can handle this event and get information about the selected element and its index.

It is important to note that according to JS standards, months are numbered starting from zero, where January corresponds to the zeroth month and December to the eleventh.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

type: 'month',

onClickMonth(self) {

console.log(self.context.selectedMonth);

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Action Handlers**

**Handling Arrow Clicks**

When any of the arrows are clicked, an event occurs to switch the month or year in the calendar. This event can be used according to your needs.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

onClickArrow(self) {

console.log(self.context.selectedYear, self.context.selectedMonth);

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Action Handlers**

**Handling Year Click in Year Selection**

Just like selecting a month, you can select a year by clicking on the year header in the calendar.

When a year is clicked from the list, you can get information about the selected element that was clicked, as well as the year number.

Just like selecting a month, you can select a year by clicking on the year header in the calendar.

When a year is clicked from the list, you can get information about the selected element that was clicked, as well as the year number.

**TS Live Example**

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

type: 'year',

onClickYear(self) {

console.log(self.context.selectedYear);

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Handling Day of Week and Week Number Clicks**

**Day of the Week**

You can intercept a click on a day of the week and, for example, select all days of the month that correspond to that day of the week.

import { Calendar, type FormatDateString, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectionDatesMode: 'multiple',

onClickWeekDay(self, day, dateEls) {

const selectedDates = dateEls.map((dateEl) => dateEl.dataset.vcDate) as FormatDateString[];

self.set({ selectedDates }, { dates: true });

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Week Number**

You can display week numbers in the calendar using the enableWeekNumbers parameter and handle clicks on them. Having information about the dates in the selected week, you can easily select these dates in the same way.

**TS Live Example**

import { Calendar, type FormatDateString, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

enableWeekNumbers: true,

selectionDatesMode: 'multiple-ranged',

onClickWeekNumber(self, number, year, dateEls) {

const selectedDates = dateEls.map((dateEl) => dateEl.dataset.vcDate) as FormatDateString[];

self.set({ selectedDates }, { dates: true });

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Action Handlers**

**Getting and Modifying Each Day**

With access to each day in the calendar, you can perform various operations, add additional information, or make changes to each day.

For example, you can add a random cost or value to each day.

onCreateDateEls(self, dateEl) {

const randomBoolean = Math.random() < 0.5;

if (!randomBoolean) return;

const randomPrice = Math.floor(Math.random() \* (999 - 100 + 1) + 100);

const btnEl = dateEl.querySelector('[data-vc-date-btn]') as HTMLButtonElement;

const day = btnEl.innerText;

btnEl.style.flexDirection = 'column';

btnEl.innerHTML = `

<span>${day}</span>

<span style="font-size: 8px;color: #8BC34A;">$${randomPrice}</span>

`;

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Action Handlers**

**Selecting and Changing Time**

By activating the selectionTimeMode parameter, you gain the ability to automatically receive the necessary data with each change of time.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectionTimeMode: 12,

onChangeTime(self) {

console.log(self.context.selectedTime);

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Additional Features**

**Layouts**

The calendar provides a convenient way to customize the HTML markup using the layouts parameter. This allows you to add your own elements, such as buttons or any other HTML element, to the calendar.

layouts takes the type of the calendar as the key and a string as the value.

In the following example, the calendar header is customized for type: 'default', and a regular button is added inside the calendar.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

layouts: {

default: `

<div class="vc-header" data-vc="header" role="toolbar" aria-label="Calendar Navigation">

<div class="vc-header\_\_content" data-vc-header="content">

<#Year /> | <#Month />

</div>

<#ArrowPrev />

<#ArrowNext />

</div>

<div class="vc-wrapper" data-vc="wrapper">

<#WeekNumbers />

<div class="vc-content" data-vc="content">

<#Week />

<#Dates />

<#DateRangeTooltip />

</div>

</div>

<#ControlTime />

<button type="button">I am a button</button>

`,

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

Now, let's use the inputMode: true parameter. We will add a button that will hide the calendar when clicked.

**TS Live Example**

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

inputMode: true,

onChangeToInput(self) {

if (!self.context.inputElement) return;

if (self.context.selectedDates[0]) {

self.context.inputElement.value = self.context.selectedDates[0];

} else {

self.context.inputElement.value = '';

}

},

onInit(self) {

const handleClickMainElement = (e: MouseEvent) => {

if ((e.target as HTMLElement).closest('#btn-close')) {

self.hide();

}

};

self.context.mainElement.addEventListener('click', handleClickMainElement);

return () => self.context.mainElement.removeEventListener('click', handleClickMainElement);

},

layouts: {

default: `

<div class="vc-header" data-vc="header" role="toolbar" aria-label="Calendar Navigation">

<#ArrowPrev />

<div class="vc-header\_\_content" data-vc-header="content">

<#Month />

<#Year />

</div>

<#ArrowNext />

</div>

<div class="vc-wrapper" data-vc="wrapper">

<#WeekNumbers />

<div class="vc-content" data-vc="content">

<#Week />

<#Dates />

<#DateRangeTooltip />

</div>

</div>

<#ControlTime />

<button id="btn-close" type="button">Close</button>

`,

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Additional Features**

**Popups and Tooltips**

**Popups**

The calendar allows you to add popups with information for any day, which will be displayed when hovering over that day.

In the provided example, a specific day is highlighted using a CSS modifier, and information is added to the popup.

Additional details about popups can be found in the reference guide.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectedMonth: 6,

selectedYear: 2024,

popups: {

'2024-07-03': {

modifier: 'bg-sponsor',

html: `

<div>

💖 Support the project: <a href="https://buymeacoffee.com/uvarov" rel="noopener noreferrer" target="\_blank">Vanilla Calendar Pro</a>

</div>

`,

},

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Tooltips**

Tooltips can be used when the selectionDatesMode parameter is set to 'multiple-ranged'. Using onCreateDateRangeTooltip, you can create a fully customized tooltip.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectionDatesMode: 'multiple-ranged',

onCreateDateRangeTooltip(self) {

const createRow = (title: string, value: string) =>

`<div style="text-align: left; white-space: nowrap">

<span>${title}</span>

<b>${value}</b>

</div>`;

return `

${createRow('Start:', self.context.selectedDates[0])}

${self.context.selectedDates[1] ? createRow('End:', self.context.selectedDates[1]) : ''}

`;

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Additional Features**

**Styles**

All CSS classes used in the calendar are variables that can be customized by replacing them with your own values.

When replacing CSS classes with your own, keep in mind that you will need to create and style this class in your own CSS.

Below is an example of replacing the class for the arrows with your own. A full list of classes can be found in the reference guide.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

styles: {

arrowPrev: 'arrow-smile',

},

};

const calendar = new Calendar('#calendar', options);

calendar.init();

// Add to your css and uncomment:

// button.arrow-smile::before {

// background: url("data:image/svg+xml;utf8,<svg xmlns='http://www.w3.org/2000/svg' viewport='0 0 24 24' style='fill:black;font-size:24px;'><text y='90%' x='0'>😊</text></svg>");

// transform: rotate(0);

// transition: transform 0.2s;

// }

// button.arrow-smile:hover::before {

// transform: rotate(180deg);

// }

**Additional Features**

**Themes**

The calendar supports custom themes and by default has light and dark themes.

If the themeAttrDetect parameter is set to false, the theme will be determined by the user's system settings or the selectedTheme parameter.

The calendar can automatically detect and track the site's theme based on the set tag and attribute. Additional information about this parameter can be found in the reference guide.

If your site supports only one theme or you want to customize the appearance of the calendar to your liking, you can explicitly select one of the available themes.

The example below demonstrates the forced use of the dark theme:

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectedTheme: 'dark',

};

const calendar = new Calendar('#calendar', options);

calendar.init();

And here is the same example, but using the light theme:

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/index.css';

const options: Options = {

selectedTheme: 'light',

};

const calendar = new Calendar('#calendar', options);

calendar.init();

As described above, you can use your own themes, create them yourself, or import them from the calendar if they exist.

import { Calendar, type Options } from 'vanilla-calendar-pro';

import 'vanilla-calendar-pro/styles/layout.css';

import 'vanilla-calendar-pro/styles/themes/slate-light.css';

const options: Options = {

selectedTheme: 'slate-light',

};

const calendar = new Calendar('#calendar', options);

calendar.init();

**Components for Libraries**

**React Component**

If you are not using TypeScript, use the .jsx extension instead of .tsx and remove the CalendarProps interface from the component.

For demonstration purposes, let's consider the simplest React component for Vanilla Calendar Pro. Create a file named VanillaCalendar.tsx and copy the following code into it:

import { useEffect, useRef, useState } from 'react';import { Options, Calendar } from 'vanilla-calendar-pro'; import 'vanilla-calendar-pro/styles/index.css'; interface CalendarProps extends React.HTMLAttributes<HTMLDivElement> { config?: Options,} function VanillaCalendar({ *config*, ...*attributes* }: CalendarProps) { const ref = useRef(null); const [calendar, setCalendar] = useState<Calendar | null>(null); useEffect(() => { if (!ref.current) return; setCalendar(new Calendar(ref.current, config)); }, [ref, config]) useEffect(() => { if (!calendar) return; calendar.init() }, [calendar]) return ( <div {...attributes} *ref*={ref}></div> )} export default VanillaCalendar;

Then, import the created VanillaCalendar component into your React application where you plan to display the calendar.

tsxCopy

import VanillaCalendar from './VanillaCalendar';

Use the created component.

tsxCopy

*// ...*<VanillaCalendar />*// ...*

The VanillaCalendar component can accept any HTML attributes supported by the <div> tag, as well as the config parameter for configuring the calendar.

tsxCopy

*// ...*<VanillaCalendar *config*={{ type: 'multiple', }} *className*="thisIsMyClass" />*// ...*

**Components for Libraries**

**Vue Component**

To demonstrate, let's create a simple Vue component for Vanilla Calendar Pro. Create a file named VanillaCalendar.vue and copy the following code into it:

vueCopy

<script setup lang="ts">import { onMounted, ref, useAttrs } from 'vue';import { Calendar, Options } from 'vanilla-calendar-pro';import 'vanilla-calendar-pro/styles/index.css' const calendarRef = ref(null);const attributes = useAttrs();const { config } = defineProps<{ config?: Options }>(); onMounted(() => { if (!calendarRef.value) return; const calendar = new Calendar(calendarRef.value, config); calendar.init();});</script> <template> <div v-bind="attributes" ref="calendarRef"></div></template>

Then import the created VanillaCalendar component into your Vue application where you want to display the calendar.

vueCopy

<script setup lang="ts">*// ...*import VanillaCalendar './VanillaCalendar.vue';*// ...*</script>

Use the created component.

vueCopy

<template> *<!-- -->* <VanillaCalendar /> *<!-- -->*</template>

The VanillaCalendar component can accept any HTML attributes supported by the <div> tag, as well as the config parameter for calendar configuration.

vueCopy

<template> *<!-- -->* <VanillaCalendar :config="{ type: 'multiple' }" /> *<!-- -->*</template>